Geomajas common-gwt plug-in guide

Geomajas Developers and Geosparc

Geomajas common-gwt plug-in guide

by Geomajas Developers and Geosparc

1.0.0

Table of Contents

List of Examples

Chapter 1. Common GWT

The common GWT module is a module with common code for the two GWT based faced, the GWT face which uses the SmartGWT widget library, and the pure GWT face which is not tied to a widget library.

Some of the features include the command dispatcher, which includes some important security hooks, and a string of utility classes which can help you in your application.

1. Setting up GWT-RPC communication

Geomajas makes use of GWT RPC, which is a mechanism for passing Java objects to and from a server over standard HTTP. GWT RPC allows the client to call a server-based service through an automatically generated proxy class. Synchronous or asynchronous calls can be made to the server through a very efficient protocol on top of HTTP. The service endpoint is defined by configuring a standard servlet mapping or an MVC controller, which is more in line with Spring MVC practices.

Because the GWT module name is the first part of the service endpoint URL, the MVC controller should also be mapped to this module name. This requires the following addition to the web.xml file:

```
<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/d/*</url-pattern>
  <url-pattern>/<GWT module name>/*</url-pattern> <!-- add this line -->
</servlet-mapping>
```

This is all that is necessary to enable the communication and should work for all GWT faces. This will also establish a fallback resource controller that ensures that static resources can be successfully retrieved through the dispatcher servlet. This is necessary because the above mapping will catch any request that has the module name as the first part in its path.

2. Command dispatcher

The command dispatcher allows you to communicate to the server using the Geomajas command interface with the Geomajas built-in support for security and localization. Using the GwtCommandDispatcher class, you can send your request objects to the server and process the response.

A simple invocation looks like this:

Example 1.1. Example use of executing a command.

```
MySuperDoItRequest commandRequest = new MySuperDoItRequest();
// .... add parameters to the request.

// Create the command, with the correct Spring bean name:
GwtCommand command = new GwtCommand(MySuperDoItRequest.COMMAND);
command.setCommandRequest(commandRequest);

// Execute the command, and do something with the response:
GwtCommandDispatcher.getInstance().execute(command, new AbstractCommand

    public void execute(MySuperDoItResponse response) {
        // Command returned successfully. Do something with the result.
    }
});
```

The security will be handled automatically by the command dispatcher.

You can register a token request handler which is invoked when the command returns with an invalid token exception (or any security exception when the token is null). Once the new token is available, the command will be retried.

There are events on the dispatcher for changes in token. You can register a handler for them.

There are also events to indicate when a command is in process or not.

The dispatcher includes methods to request details about the current token and the logged i user.

3. Controller

This module also provides definitions for event controllers on the map. A controller is meant to catch mouse or touch events and react upon them. Together with the controller interface, an AbstractController class is provided as a more suitable base to start working from. Still, each face has extended this base controller interface with it's own methods (for activation and deactivation for example). Each face also has an abstract controller class to be used as a base to implement your own controllers.

3.1. Mouse and touch events

The AbstractController tries to align the mouse and touch events by providing extra methods through the MapDownHandler, MapUpHandler and MapDragHandler. The key is that both mouse events and touch events extend the same base class, HumanInputEvent. The three handlers provide the following methods:

- onDown(HumanInputEvent): Overriding this empty method will provide support for both the onMouseDown and onTouchStart events.
- onUp(HumanInputEvent): Overriding this empty method will provide support for both the onMouseUp and onTouchEnd events.
- onDrag(HumanInputEvent): Overriding this method will provide support for the onTouchMove and the onMouseMove events. The onMouseMove events though will only get here is the left mouse button is down. Simply put, this method accounts for dragging.

You can still choose to override the original mouse and/or touch handling methods if you are not interested in having both desktop and mobile support.

Tip

If you need a specific reaction to the onMouseMove in combination with dragging behaviour, you can still override the onMouseMove, and have it also call the super implementation (from AbstractController).

3.2. Event utility methods

The controller interface also provides a few utility methods that operate on HumanInputEvents (again to support both mouse and touch events). This includes getting the target HTML element from an event or getting the event location in the required render space (pixels or map CRS).

4. Getting configuration

There is a special service for getting the client configuration from the server. This is the ClientConfigurationService class. It serves as a central access point for your application configuration and limits the number of request for configuration information to the server.

It is specifically built to be extensible. You con define how the application configuration is obtained from the server. This allows you to replace the invocation of the GetConfiguration command by something which also contains application specific information. Have a look at the staticsecurity example application for an example how this can be used.

5. Utility classes

Several utility classes have been provided.

5.1. Log

This is a simple utility class which allows you to log data.

You can force logging to the server log or use the usual log levels. The normal logging uses commons logging. You can configure in your .gwt.xml file how this should be treated, see http://code.google.com/webtoolkit/doc/latest/DevGuideLogging.html. The error and warn levels will also log to the server.

5.2. HTML building utilities

- Dom: includes browser identification tests and constants and helpers for various DOM related manipulations.
- Html: constants for the various HTML tag and attribute names.
- HtmlBuilder: helper for building pieces of HTML. Usually used in combination with the constants from Html.

5.3. General utilities

- AttributeUtil: helper to build attribute objects.
- EqualsUtil: helps to determine if two objects are equal in a null-safe way.
- StringUtil: some string handling methods.